NumPy简介

NumPy（Numerical Python）是 Python 进行科学计算的一个扩展库，提供了大量的函数和操作，主要用于对多维数组执行计算，它比 Python 自身的嵌套列表结构要高效的多。

NumPy 数组和 Python 列表的主要区别：

- 数组会对元素的数据类型做统一，而列表不会。
- 数组创建后具有固定大小，而列表由于内存自动管理，可动态调整。

创建数组

np.array(object, dtype=None)

- object：array_like，类似于数组的对象。如果object是标量，则返回包含object的0维数组
- dtype：data-type，数据类型。如果没有给出，会从输入数据推断数据类型
- 创建一个数组对象并返回（ndarray实例对象）

DTYPE常用值	描述
np.int8	有符号整数（1个字节）
np.int16	有符号整数（2个字节）
np.int32	有符号整数（4个字节）

DTYPE常用值	描述
np.int64	有符号整数（8个字节）
np.uint8	无符号整数（1个字节）
np.uint16	无符号整数（2个字节）
np.uint32	无符号整数（4个字节）
np.uint64	无符号整数（8个字节）
np.float16	半精度浮点数（2个字节）
np.float32	单精度浮点数（4个字节）
np.float64	双精度浮点数（8个字节）

NDARRAY常用属性	描述
ndarray.ndim	秩，即轴的数量或维度的数量
ndarray.shape	数组的形状
ndarray.size	数组中数据的总个数
ndarray.dtype	数组中的数据类型
ndarray.itemsize	数组中的数据大小，以字节为单位

```
import numpy as np



num = 789
arr = np.array(num)
print(num)
print(arr)
print(type(num))
print(type(arr))



lst = [6, 7, 1, 0, 9, 8]
```

```
arr = np.array(lst)
print(lst)
print(arr)


lst = [[6, 7, 1], [0, 9, 8]]
arr = np.array(lst)
print(lst)
print(arr)
print(arr.ndim)
print(arr.dtype)
print(arr.itemsize)
print(arr.shape)
print(arr.size)


lst = [[[6, 7], [1, 0], [9, 8]]]
arr = np.array(lst)
print(lst)
print(arr)
```

np.arange([start,] stop[, step])

- 返回给定区间内的均匀间隔值构成的数组

```
import numpy as np



print(np.arange(3))
print(np.arange(3.0))
print(np.arange(3, 7))
print(np.arange(3, 7, 2))
print(np.arange(7, 3, -2))
print(np.arange(3, 7, 0.5))
```

np.linspace(start, stop, num=50, dtype=None)

- num：生成的样本数量
- dtype：默认自动推断数据类型，推断出的 dtype 永远不会是整数；即使参数会产生一个整数数组，也会选择 np.float64
- 把给定区间分成 num 个均匀间隔的样本，构成数组并返回

```
import numpy as np



print(np.linspace(1, 50))
print(np.linspace(1, 10, num=10))
print(np.linspace(1, 10, num=10, dtype=np.int32))
```

基本运算

数组的算术运算和比较运算为逐元素操作

```
import numpy as np


a = np.array([[1, 2], [3, 4], [5, 6]])
print(a + 2)
print(a - 2)
print(a * 2)
print(a / 2)
print(a < 4)
print(a > 3)

b = np.array([[2, 2], [2, 1], [1, 1]])
print(a + b)
print(a - b)
print(a * b)
print(a / b)
print(a < b)
print(a > b)
```

广播机制

- 后缘维度相同或者不同的维度有1，可以广播

```
import numpy as np


a = np.arange(24).reshape((2, 3, 4))
b = np.arange(12).reshape((3, 4))
```

```
c = np.arange(4).reshape((1, 4))
d = np.arange(4).reshape(4)
e = np.arange(12).reshape((1, 3, 4))
f = np.arange(6).reshape((2, 3, 1))
g = np.arange(2).reshape((2, 1, 1))
h = np.arange(2).reshape((1, 2, 1, 1))
i = np.arange(10).reshape((5, 2, 1, 1))

print((a + b).shape)
print((a + c).shape)
print((a + d).shape)
print((a + e).shape)
print((a + f).shape)
print((a + g).shape)
print((a + h).shape)
print((a + i).shape)
```

索引和切片

数组除了支持Python序列的索引和切片操作以外，还可以针对各个轴进行索引和切片操作。

序列索引和切片

```
import numpy as np
```

```
lst = [6, 8, 9, 1, 3]
arr = np.array(lst)
print(lst)
print(arr)

item_lst = lst[2]
part_lst = lst[2:3]
item_arr = arr[2]
part_arr = arr[2:3]  # 返回视图
print(item_lst)
print(part_lst)
print(item_arr)
print(part_arr)

lst[2] = 99
arr[2] = 99
print(item_lst)
print(part_lst)
print(item_arr)
print(part_arr)  # 动态性
```

数组针对各个轴的索引和切片

```
import numpy as np


lst = [[[6, 7, 5, 1],
        [2, 9, 8, 0],
        [3, 4, 2, 8]],

       [[4, 5, 2, 3],
        [2, 9, 7, 1],
```

```
        [9, 5, 6, 7]]]

arr = np.array(lst)  # shape: (2, 3, 4)
print(lst[1][0][2])
print(arr[1][0][2])
print(arr[1, 0, 2])

print(lst[1:2][:1])
print(arr[1:2][:1])
print(arr[1:2, :1])

print(lst[1][::2][0])
print(arr[1][::2][0])
print(arr[1, ::2, 0])
```

数组的高阶索引

- 把整数列表或者bool数组作为索引

```
import numpy as np



x = np.arange(24).reshape((3, 2, 4))
print(x)

# 可以理解为x[2], x[0], x[0]构成一个更高维度的数组
print(x[[2, 0, 0]])

# 可以理解为x[2, 0], x[0, 0], x[1, 1]构成一个更高维度的数组
print(x[[2, 0, 1], [0, 0, 1]])
```

```
# 可以理解为x[2, 0, 1], x[0, 0, 2], x[1, 1, 3]构成一个更高维度的数组
print(x[[2, 0, 1], [0, 0, 1], [1, 2, 3]])

# 基本索引和高阶索引组合时, 会发生广播, 下面三个是等价的
print(x[0, [0, 0, 1], [1, 2, 3]])
print(x[[0], [0, 0, 1], [1, 2, 3]])
print(x[[0, 0, 0], [0, 0, 1], [1, 2, 3]])

# 下面三个也是等价的
print(x[0, [0, 0, 1], 2])
print(x[[0], [0, 0, 1], [2]])
print(x[[0, 0, 0], [0, 0, 1], [2, 2, 2]])

# 切片在高阶索引一侧, 按照轴的顺序定shape即可
print(x[::2, [0, 0, 1], [3, 0, 2]])  # shape: (2, 3)
print(x[[2, 0, 1], [1, 0, 1], ::2])  # shape: (3, 2)

# 切片两侧都有高阶索引时, 定shape时高阶索引在前, 切片在后
print(x[[2, 0, 1], 1:, [3, 0, 2]])  # shape: (3, 1)
```

```
import numpy as np


x = np.arange(24).reshape((3, 2, 4))
print(x)

""" 如果需要利用bool索引对标量进行操作, 也就是针对最后一个维度, 只需要创建一个shape为(3, 2, 4)的bool索引即可 """
bool_list = [[[True, False, True, False],
              [False, True, False, True]],
             [[True, False, True, False],
              [False, True, False, True]],
```

```
                [[True, False, True, False],
                 [False, True, False, True]]

print(x[np.array(bool_list)])

# x > 13 得到一个shape为(3, 2, 4)的bool数组
print(x[x > 13])

""" 如果需要利用bool索引对1轴进行操作，只需要创建一个shape为(3, 2)的bool索引即可 """
bool_list = [[True, False],
             [False, True],
             [True, False]]

print(x[np.array(bool_list)])

""" 如果需要利用bool索引对0轴进行操作，只需要创建一个shape为(3,)的bool索引即可 """
bool_list = [True, False, True]
print(x[np.array(bool_list)])
```

常用操作

---

np.reshape(arr_like, newshape)

- 保证 size 不变，在不更改数据的情况下为数组赋予新的形状
- newshape如果是整数，则结果将是该长度的1-D数组
- newshape的一个形状维度可以是-1，值将自行推断

```
import numpy as np


arr1 = np.arange(6).reshape((2, 1, 3))
arr2 = np.reshape(arr1, 6)
arr3 = np.reshape(arr1, -1)
arr4 = np.reshape(arr1, (-1,))
print(arr2)  # [0 1 2 3 4 5]
print(arr3)  # [0 1 2 3 4 5]
print(arr4)  # [0 1 2 3 4 5]

arr1 = np.arange(24)
arr2 = np.reshape(arr1, (2, 2, -1, 2))
print(arr2.shape)  # (2, 2, 3, 2)
```

ndarray.flatten()

- 返回扁平化到一维的数组

```
import numpy as np


a = np.array([[1,2], [3,4]])
print(a.flatten())
```

ndarray.T

- 转置数组

```
import numpy as np


a = np.array([[1, 2, 3], [4, 5, 6]])
print(a)
print(a.T)

a = np.arange(24).reshape((2, 3, 4))
print(a.T.shape)
```

np.swapaxes(arr_like, axis1, axis2)

- 交换数组的两个轴

```
import numpy as np


a = np.array([[1, 2, 3], [4, 5, 6]])
print(a)
print(np.swapaxes(a, 0, 1))


a = np.arange(24).reshape((2, 3, 4))
print(np.swapaxes(a, 0, 2).shape)
```

np.transpose(arr_like, axes=None)

- 通过axes参数排列数组的shape，axes没有指定，默认为转置

```
import numpy as np


a = np.array([[1, 2, 3], [4, 5, 6]])
print(a)
print(np.transpose(a))


a = np.arange(24).reshape((2, 3, 4))
print(np.transpose(a, (1, 0, 2)).shape)
```

np.concatenate(arrays, axis=0)

- arrays：Sequence[ArrayLike]
- 沿现有轴连接一系列数组，如果axis为None，则数组在使用前会被扁平化

```
import numpy as np


a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6]])
print(np.concatenate((a, b), axis=0))
print(np.concatenate((a, b.T), axis=1))
print(np.concatenate((a, b), axis=None))
```

np.stack(arrays, axis=0)

- arrays：Sequence[ArrayLike]
- 沿新轴连接一系列数组

```
import numpy as np


a1 = np.arange(6).reshape((2, 3))
a2 = np.arange(10, 16).reshape((2, 3))
a3 = np.arange(20, 26).reshape((2, 3))
a4 = np.arange(30, 36).reshape((2, 3))
print(np.stack((a1, a2, a3, a4)).shape)
print(np.stack((a1, a2, a3, a4), axis=1).shape)
print(np.stack((a1, a2, a3, a4), axis=2).shape)
```

np.dot(a, b)

- 两个数组的点积

```
import numpy as np


a = [1, 2, 3]
b = [1, 0, 2]
print(np.dot(a, b))

a = [[1, 0], [0, 1]]
b = [[4, 1], [2, 2]]
print(np.dot(a, b))
```

np.matmul(x1, x2)
@操作符也可表示

- 两个数组的矩阵乘积

```
import numpy as np



a = np.array([[1, 0],
              [0, 1]])
b = np.array([[4, 1],
              [2, 2]])
print(np.matmul(a, b))
print(a @ b)
```

np.greater(x1, x2)

- x1，x2的形状必须相同，或者可以广播
- 按元素判断 x1 > x2 的结果

```
import numpy as np



print(np.greater([4, 2], [2, 2]))

a = np.array([[4, 2], [3, 1]])
b = np.array([[2, 2]])
print(np.greater(a, b))
print(a > b)
```

np.greater_equal(x1, x2)

- x1，x2的形状必须相同，或者可以广播
- 按元素判断 x1 >= x2 的结果

```
import numpy as np


print(np.greater_equal([4, 2], [2, 2]))

a = np.array([[4, 2], [3, 1]])
b = np.array([[2, 2]])
print(np.greater_equal(a, b))
print(a >= b)
```

np.less(x1, x2)

- x1，x2的形状必须相同，或者可以广播
- 按元素判断 x1 < x2 的结果

```
import numpy as np


print(np.less([4, 2], [2, 2]))

a = np.array([[4, 2], [3, 1]])
b = np.array([[2, 2]])
print(np.less(a, b))
print(a < b)
```

np.less_equal(x1, x2)

- x1，x2的形状必须相同，或者可以广播
- 按元素判断 x1 <= x2 的结果

```
import numpy as np



print(np.less_equal([4, 2], [2, 2]))

a = np.array([[4, 2], [3, 1]])
b = np.array([[2, 2]])
print(np.less_equal(a, b))
print(a <= b)
```

np.equal(x1, x2)

- x1，x2的形状必须相同，或者可以广播
- 按元素判断 x1 == x2 的结果

```
import numpy as np



print(np.equal([4, 2], [2, 2]))

a = np.array([[4, 2], [3, 1]])
b = np.array([[2, 2]])
print(np.equal(a, b))
print(a == b)
```

np.not_equal(x1, x2)

- x1，x2的形状必须相同，或者可以广播
- 按元素判断 x1 != x2 的结果

```
import numpy as np



print(np.not_equal([4, 2], [2, 2]))

a = np.array([[4, 2], [3, 1]])
b = np.array([[2, 2]])
print(np.not_equal(a, b))
print(a != b)
```

np.sin(x)

- x：角度（弧度值）
- 正弦函数

```
import numpy as np



print(np.sin(np.pi/2))
print(np.sin(np.array((0, 30, 90)) * np.pi / 180))
```

np.cos(x)

- x：角度（弧度值）
- 余弦函数

```
import numpy as np



print(np.cos(np.pi/2))
print(np.cos(np.array((0, 60, 90)) * np.pi / 180))
```

np.tan(x)

- x：角度（弧度值）
- 正切函数

```
import numpy as np


print(np.tan(-np.pi))
print(np.tan(np.array((0, 180)) * np.pi / 180))
```

np.arcsin(x)

- 反正弦函数

```
import numpy as np


print(np.arcsin(1))
print(np.arcsin(np.array([0.5, -0.5])))
```

np.arccos(x)

- 反余弦函数

```
import numpy as np


print(np.arccos(-1))
print(np.arccos(np.array([0.5, 1])))
```

np.arctan(x)

- 反正切函数

```
import numpy as np



print(np.arctan(1))
print(np.arctan(np.array([0, -1])))
```

np.floor(x)

- 返回 x 的底限

```
import numpy as np



a = np.array([-1.7, -1.5, -0.2, 0.2, 1.5, 1.7, 2.0])
print(np.floor(a))
```

np.ceil(x)

- 返回 x 的上限

```
import numpy as np



a = np.array([-1.7, -1.5, -0.2, 0.2, 1.5, 1.7, 2.0])
print(np.ceil(a))
```

np.exp(x)

- 计算 e 的 x 幂次方

```
import numpy as np


# e的0次方、e的1次方、e的2次方
print(np.exp([0, 1, 2]))
```

np.log(x)

- 计算 x 的自然对数

```
import numpy as np


print(np.log([1, np.e, np.e**2]))
```

np.log2(x)

- 计算 x 的以 2 为底的对数

```
import numpy as np


x = np.array([1, 2, 2**4])
print(np.log2(x))
```

np.log10(x)

- 计算 x 的以 10 为底的对数

```
import numpy as np



print(np.log10([1e-15, 1000]))
```

np.max(arr_like, axis=None, keepdims=False)

- 返回沿给定轴的最大值，axis没有指定时，默认为None，表示返回所有元素的最大值

```
import numpy as np



lis = [[0, 1, 7, 3], [4, 9, 6, 2], [8, 5, 11, 10]]
arr1 = np.array(lis)
print(arr1)
print(np.max(arr1))
print(np.max(arr1, axis=0))
print(np.max(arr1, axis=1))
```

np.min(arr_like, axis=None, keepdims=False)

- 返回沿给定轴的最小值，axis没有指定时，默认为None，表示返回所有元素的最小值

```
import numpy as np


lis = [[0, 1, 7, 3], [4, 9, 6, 2], [8, 5, 11, 10]]
arr1 = np.array(lis)
print(arr1)
print(np.min(arr1))
print(np.min(arr1, axis=0))
print(np.min(arr1, axis=1))
```

np.mean(arr_like, axis=None, keepdims=False)

- 返回沿给定轴的平均值，axis没有指定时，默认为None，表示返回所有元素的平均值

```
import numpy as np


lis = [[0, 1, 7, 3], [4, 9, 6, 2], [8, 5, 11, 10]]
arr1 = np.array(lis)
print(arr1)
print(np.mean(arr1))
print(np.mean(arr1, axis=0))
print(np.mean(arr1, axis=1))
```

np.var(arr_like, axis=None, keepdims=False)

- 返回沿给定轴的方差，axis没有指定时，默认为None，表示返回所有元素的方差

```
import numpy as np


lis = [[0, 1, 7, 3], [4, 9, 6, 2], [8, 5, 11, 10]]
arr1 = np.array(lis)
print(arr1)
print(np.var(arr1))
print(np.var(arr1, axis=0))
print(np.var(arr1, axis=1))
```

np.std(arr_like, axis=None, keepdims=False)

- 返回沿给定轴的标准差，axis没有指定时，默认为None，表示返回所有元素的标准差

```
import numpy as np


lis = [[0, 1, 7, 3], [4, 9, 6, 2], [8, 5, 11, 10]]
arr1 = np.array(lis)
print(arr1)
print(np.std(arr1))
print(np.std(arr1, axis=0))
print(np.std(arr1, axis=1))
```

np.prod(arr_like, axis=None, keepdims=np._NoValue, initial=np._NoValue)

- 返回给定轴上数组元素的乘积

```
import numpy as np
```

```
# 默认的axis=None将计算输入数组中所有元素的乘积
print(np.prod([1, 2, 3, 4]))
print(np.prod([[1, 2], [3, 4]]))

print(np.prod([1, 2, 3, 4], initial=5))

print(np.prod([[1, 2], [3, 4]], axis=1))
print(np.prod([[1, 2], [3, 4]], axis=0))

print(np.prod([[1, 2], [3, 4]], axis=1,
keepdims=True))
print(np.prod([[1, 2], [3, 4]], axis=0,
keepdims=True))
```

np.sum(arr_like, axis=None, keepdims=np._NoValue, initial=np._NoValue)

- 返回给定轴上数组元素的和

```
import numpy as np


# 默认的axis=None将计算输入数组中所有元素的和
print(np.sum([1, 2, 3, 4]))
print(np.sum([[1, 2], [3, 4]]))

print(np.sum([1, 2, 3, 4], initial=5))

print(np.sum([[1, 2], [3, 4]], axis=1))
print(np.sum([[1, 2], [3, 4]], axis=0))

print(np.sum([[1, 2], [3, 4]], axis=1, keepdims=True))
```

```
print(np.sum([[1, 2], [3, 4]], axis=0, keepdims=True))
```

np.nonzero(arr_like)

- 返回非零元素的索引

```
import numpy as np


x = np.array([[3, 0, 0], [0, 4, 0], [5, 6, 0]])
print(x)
print(np.nonzero(x))
print(x[np.nonzero(x)])

a = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(a > 3)
print(np.nonzero(a > 3))
print(a[np.nonzero(a > 3)])
```

np.where(condition, x=None, y=None)

- condition : array_like, bool
- x, y：array_like，要么都传参，要么都不传
- 如果传三个参数，条件成立返回x，不成立时返回y
- 如果只传第一个参数，返回符合条件的元素的索引

```
import numpy as np


a = np.arange(10)
print(np.where(a < 5, a, 10*a))

# 当where内有三个参数时，当condition成立时返回x，当
condition不成立时返回y
print(np.where([[True, False], [True, True]], [[1, 2],
[3, 4]], [[9, 8], [7, 6]]))

# 如果只传第一个参数，返回符合条件的元素的索引
a = np.array([2, 4, 6, 8, 10])
print(np.where(a > 5))
```

np.argwhere(arr_like)

- 找出数组中按元素分组的非零元素的索引

```
import numpy as np


x = np.arange(6).reshape(2, 3)
print(x)
print(x>1)
print(np.argwhere(x>1))
```

np.maximum(x1, x2)

- 返回x1和x2逐个元素比较中的最大值

```
import numpy as np


print(np.maximum([2, 3, 4], [1, 5, 2]))
print(np.maximum([[2, 3], [4, 5]], [[1, 5], [2, 6]]))
```

np.minimum(x1, x2)

- 返回x1和x2逐个元素比较中的最小值

```
import numpy as np


print(np.minimum([2, 3, 4], [1, 5, 2]))
print(np.minimum([[2, 3], [4, 5]], [[1, 5], [2, 6]]))
```

np.argmax(arr_like, axis=None)

- 返回沿轴的最大值的索引

```
import numpy as np


a = np.arange(6).reshape(2, 3) + 10
print(a)

# 没有指定轴，则数组扁平化处理
print(np.argmax(a))

print(np.argmax(a, axis=0))
print(np.argmax(a, axis=1))
```

np.argmin(arr_like, axis=None)

- 返回沿轴的最小值的索引

```
import numpy as np



a = np.arange(6).reshape(2, 3) + 10
print(a)

# 没有指定轴，则数组扁平化处理
print(np.argmin(a))

print(np.argmin(a, axis=0))
print(np.argmin(a, axis=1))
```

np.random.normal(loc=0.0, scale=1.0, size=None)

- loc：均值（中心）
- scale：标准差
- size：输出的形状
- 返回从正态分布中抽取的随机样本

```
import numpy as np

print(np.random.normal(3, 2.5, size=(2, 4)))
```

np.random.randint(low, high=None, size=None)

- 返回从 [low, high) 离散均匀分布中抽取的随机整数

```python
import numpy as np

print(np.random.randint(2, size=10))  # 等价于下一行
print(np.random.randint(0, 2, size=10))  # 等价于上一行
print(np.random.randint(1, 4, size=(2, 3)))
```

np.random.uniform(low=0.0, high=1.0, size=None)

- 返回从 [low, high) 均匀分布中抽取的随机样本

```python
import numpy as np

print(np.random.uniform(2, size=10))  # 等价于下一行
print(np.random.uniform(0, 2, size=10))  # 等价于上一行
print(np.random.uniform(1, 4, size=(2, 3)))
```

np.random.permutation(x)

- x：int or array_like
- 如果 x 是整数，返回随机排列的 np.arange(x)
- 如果 x 是数组，只对数组的第一个维度随机排列，返回新的数组

```python
import numpy as np



print(np.random.permutation(6))

arr1 = np.array([0, 1, 2, 3, 4, 5])
print(np.random.permutation(arr1))

arr2 = np.arange(10).reshape(5, 2)
print(np.random.permutation(arr2))
```

np.random.seed([x])

- 随机数种子

```
import numpy as np



np.random.seed(3)
print(np.random.uniform(1, 2, size=4))

np.random.seed(5)
print(np.random.uniform(1, 2, size=4))

np.random.seed(3)
print(np.random.uniform(1, 2, size=4))

np.random.seed()
print(np.random.uniform(1, 2, size=4))
```